

# IS 6733

## Deep Learning on Cloud Platforms

### Lecture 7

#### Convolutional Neural Networks (CNN)

**Dr. Yuanxiong Guo**  
**Department of Information Systems and Cyber Security**



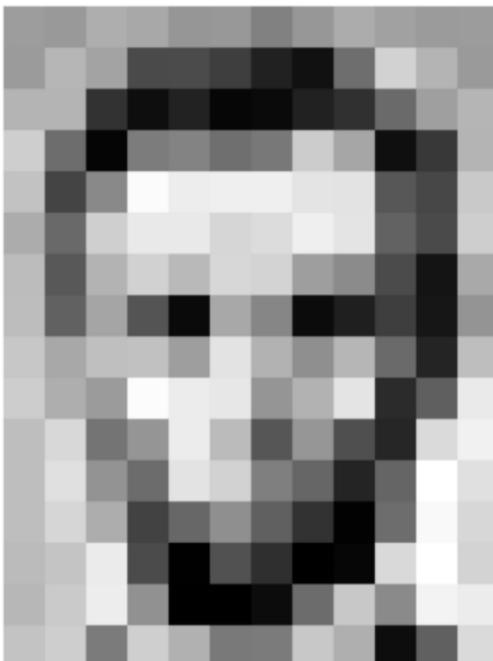
# Acknowledgement

---

- Some slides are adapted from the following courses:
  - CS231n: Convolutional Neural Networks for Visual Recognition, Stanford University
  - Alexander Amini and Ava Soleimany  
MIT 6.S191: Introduction to Deep Learning  
[IntroToDeepLearning.com](http://IntroToDeepLearning.com)

# What Computers “See”

# Images are Numbers



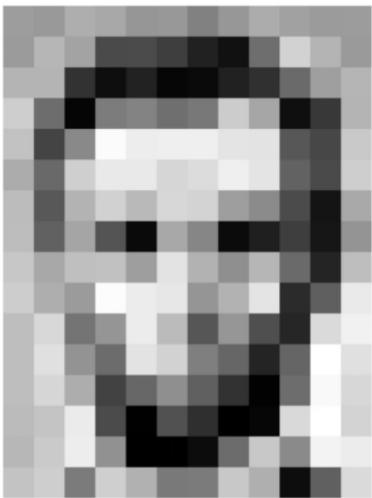
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	105	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	105	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

What the computer sees

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	105	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

An image is just a matrix of numbers [0,255]!  
i.e., 1080x1080x3 for an RGB image

# Tasks in Computer Vision



Input Image



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	26	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	195	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
196	206	123	207	177	121	123	200	175	13	96	218

Pixel Representation



Lincoln

0.8

Washington

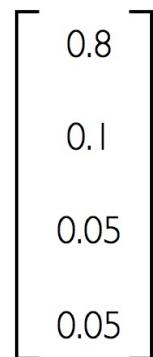
0.1

Jefferson

0.05

Obama

0.05



- **Regression:** output variable takes continuous value
- **Classification:** output variable takes class label. Can produce probability of belonging to a particular class

# High Level Feature Detection

Let's identify key features in each image category



Nose,  
Eyes,  
Mouth



Wheels,  
License Plate,  
Headlights



Door,  
Windows,  
Steps

# Manual Feature Extraction

Domain knowledge

Define features

Detect features  
to classify

Viewpoint variation



Scale variation



Deformation



Occlusion



Illumination conditions



Background clutter



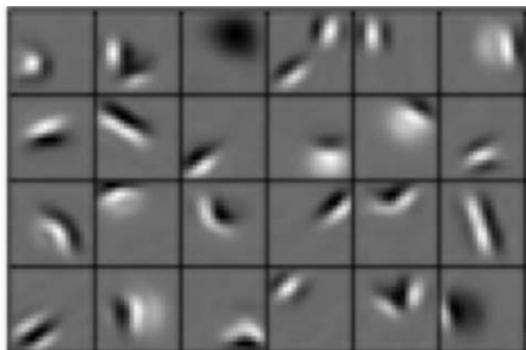
Intra-class variation



# Learning Feature Representations

Can we learn a **hierarchy of features** directly from the data instead of hand engineering?

Low level features



Edges, dark spots

Mid level features



Eyes, ears, nose

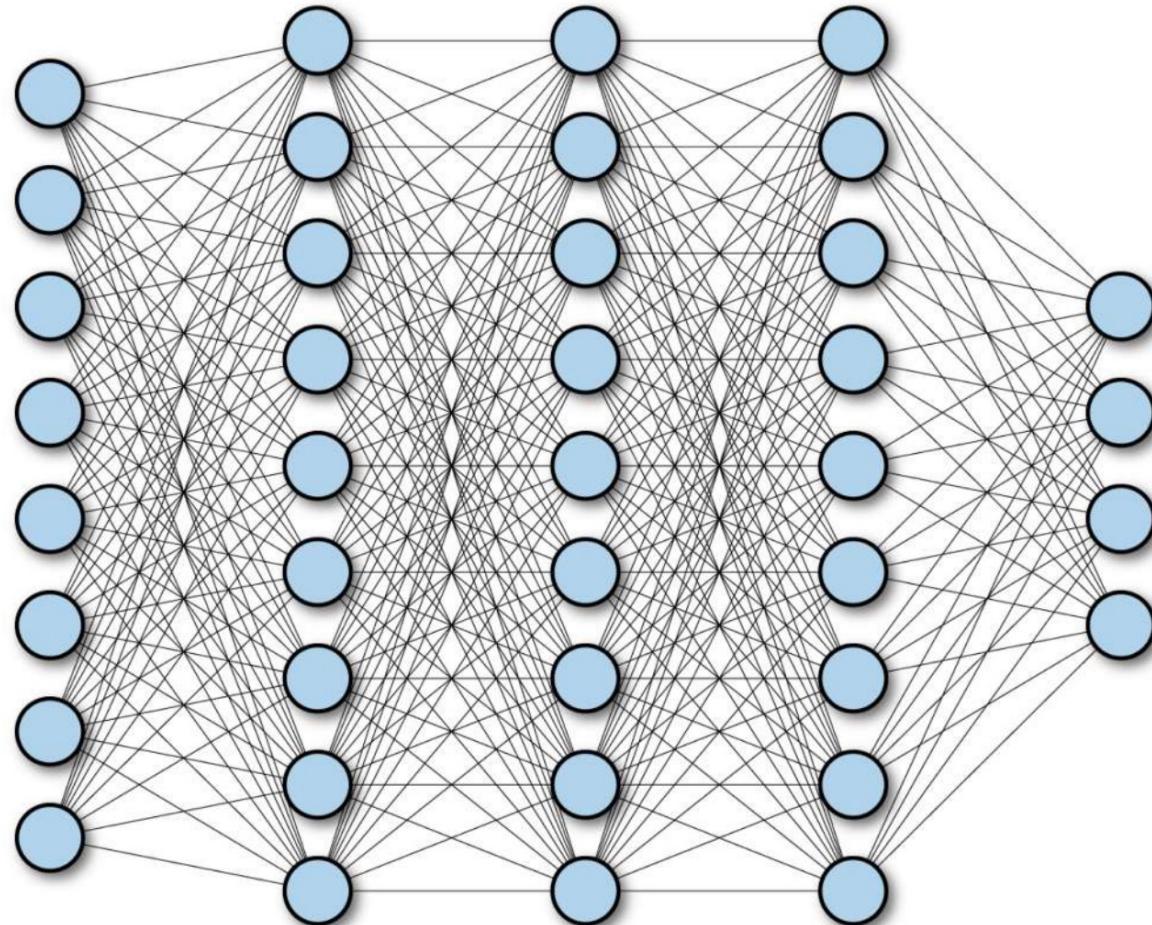
High level features



Facial structure

# CNN Basics

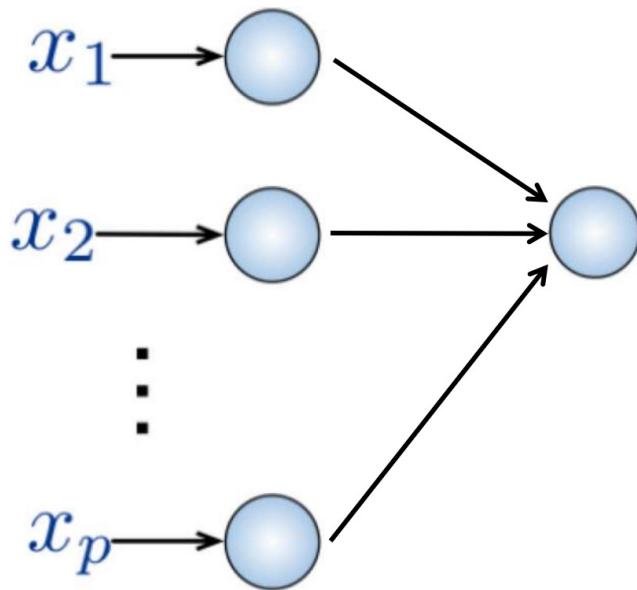
# Fully Connected Neural Network



# Fully Connected Neural Network

## Input:

- 2D image
- Vector of pixel values



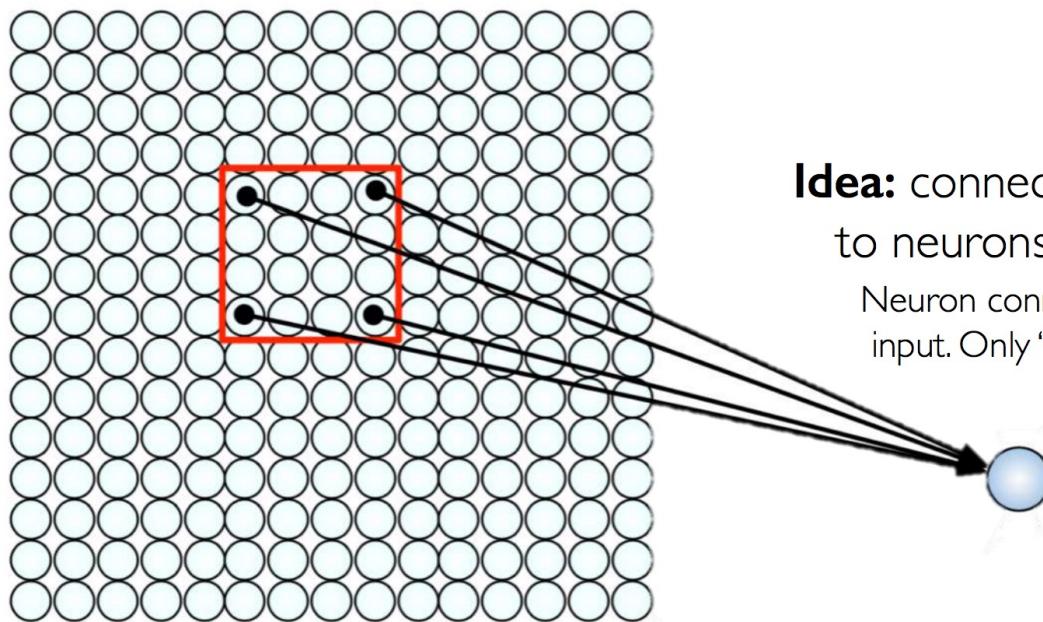
## Fully Connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

How can we use **spatial structure** in the input to inform the architecture of the network?

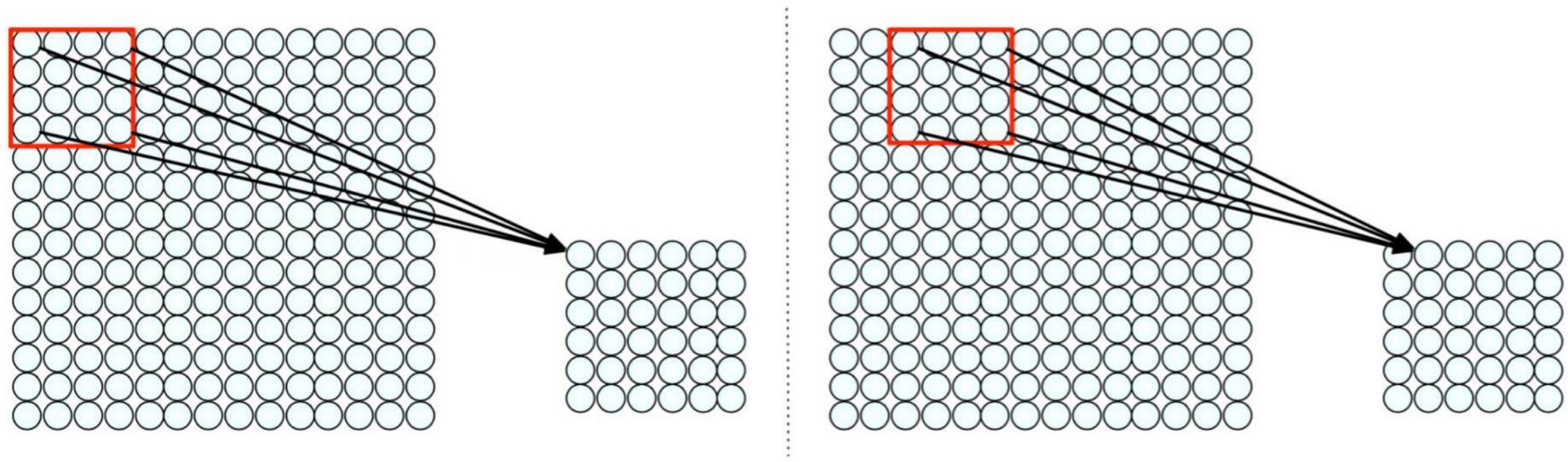
# Using Spatial Structure

**Input:** 2D image.  
Array of pixel values



**Idea:** connect patches of input  
to neurons in hidden layer.  
Neuron connected to region of  
input. Only “sees” these values.

# Using Spatial Structure



Connect patch in input layer to a single neuron in subsequent layer.

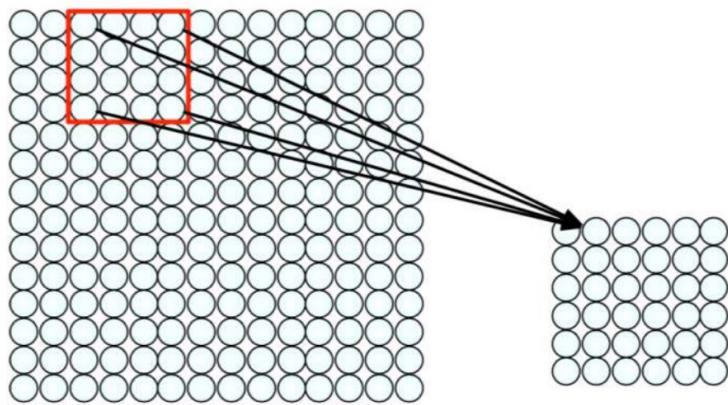
Use a sliding window to define connections.

How can we **weight** the patch to detect particular features?

# Applying Filters to Extract Features

- 1) Apply a set of weights – a filter – to extract **local features**
- 2) Use **multiple filters** to extract different features
- 3) Spatially **share** parameters of each filter  
(features that matter in one part of the input should matter elsewhere)

# Feature Extraction with Convolution



- Filter of size  $4 \times 4$  : 16 different weights
- Apply this same filter to  $4 \times 4$  patches in input
- Shift by 2 pixels for next patch

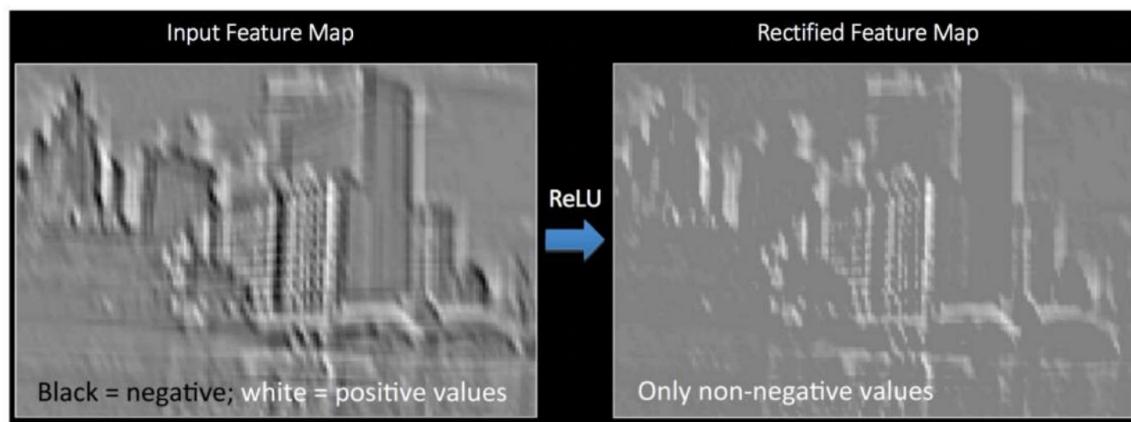
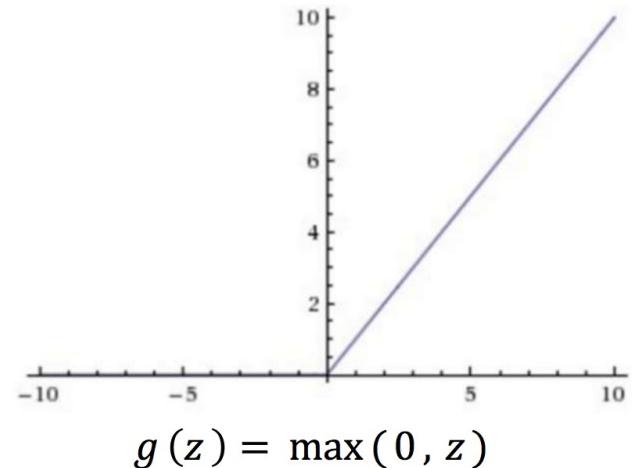
This “patchy” operation is **convolution**

- 1) Apply a set of weights – a filter – to extract **local features**
- 2) Use **multiple filters** to extract different features
- 3) **Spatially share** parameters of each filter

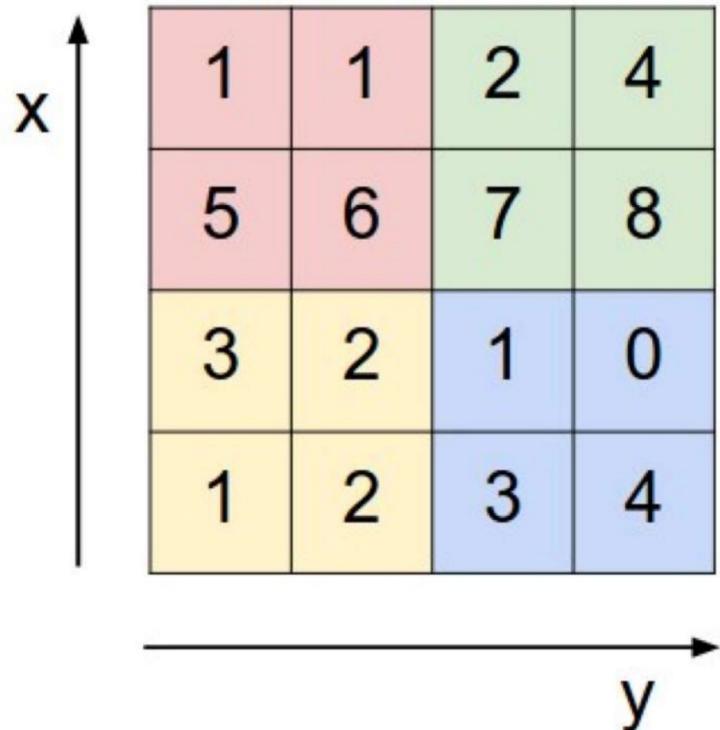
# Introducing Non-Linearity

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation!**

Rectified Linear Unit (ReLU)



# Pooling



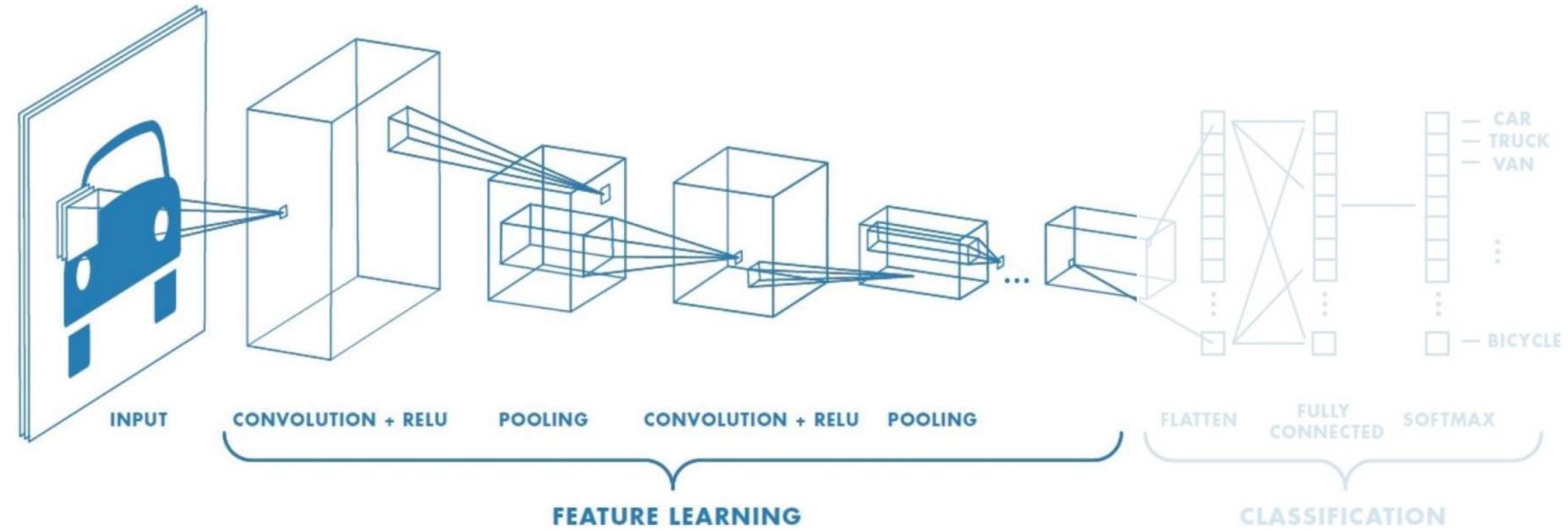
max pool with 2x2 filters  
and stride 2



6	8
3	4

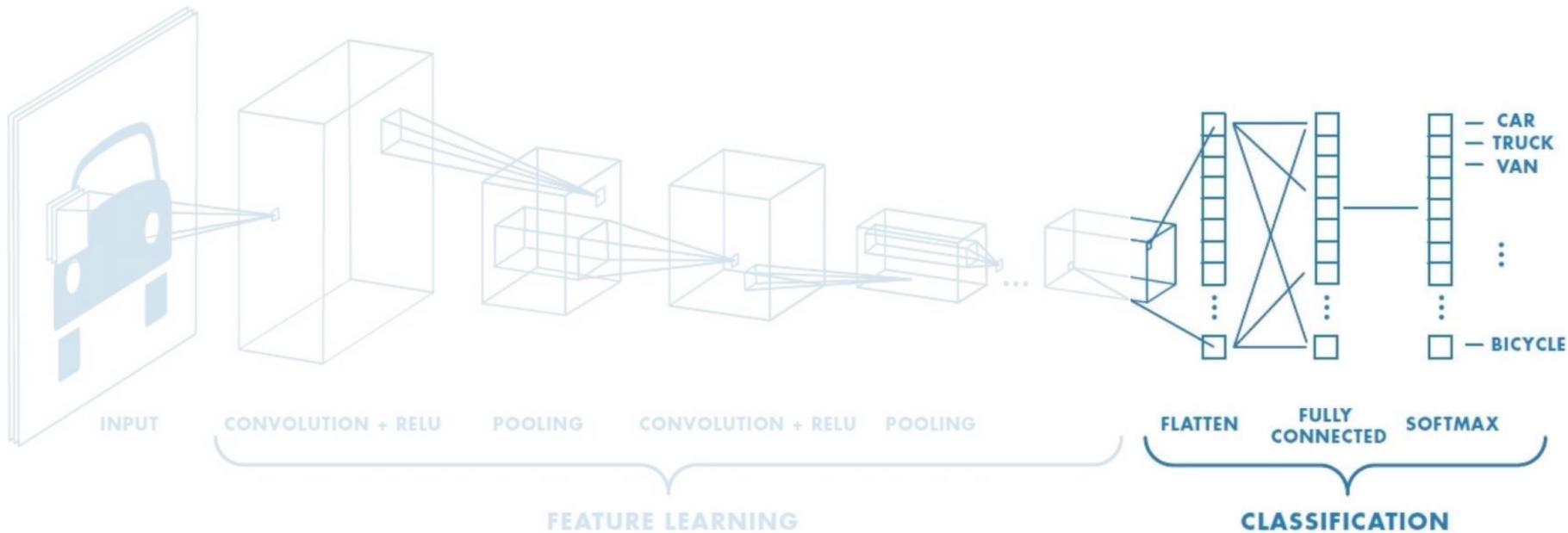
- 1) Reduced dimensionality
- 2) Spatial invariance

# CNNs for Classification: Feature Learning



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

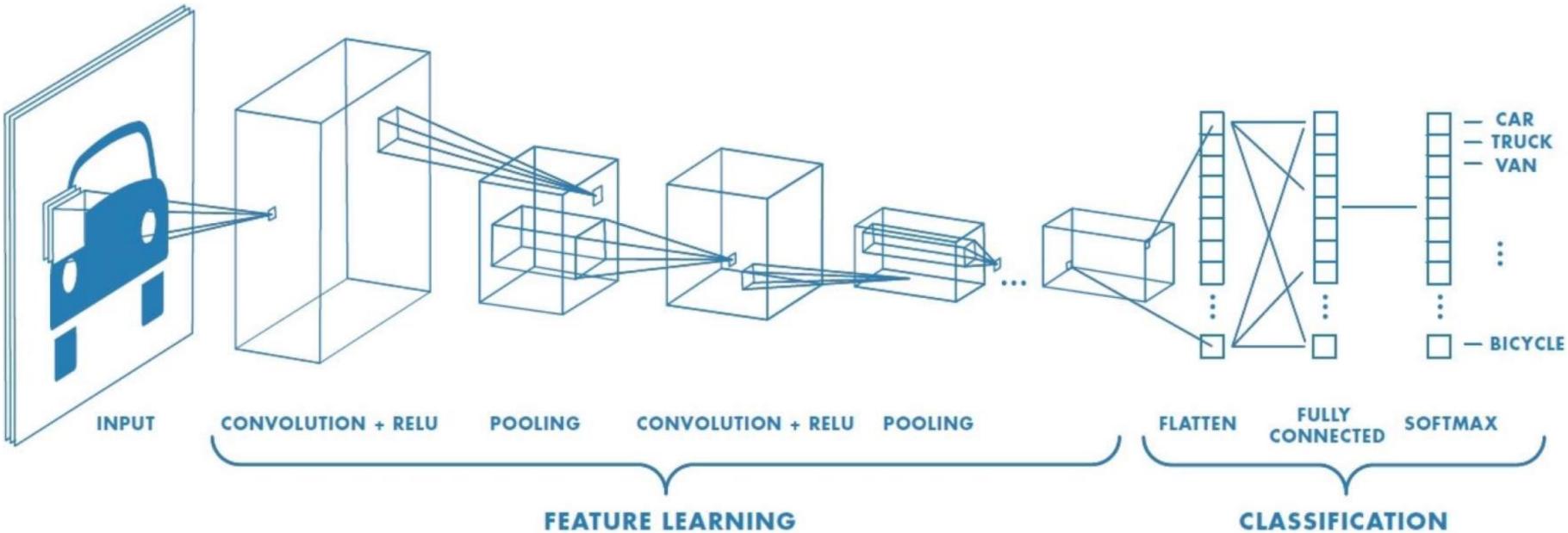
# CNNs for Classification: Class Probabilities



- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

# CNNs: Training with Backpropagation



Learn weights for convolutional filters and fully connected layers

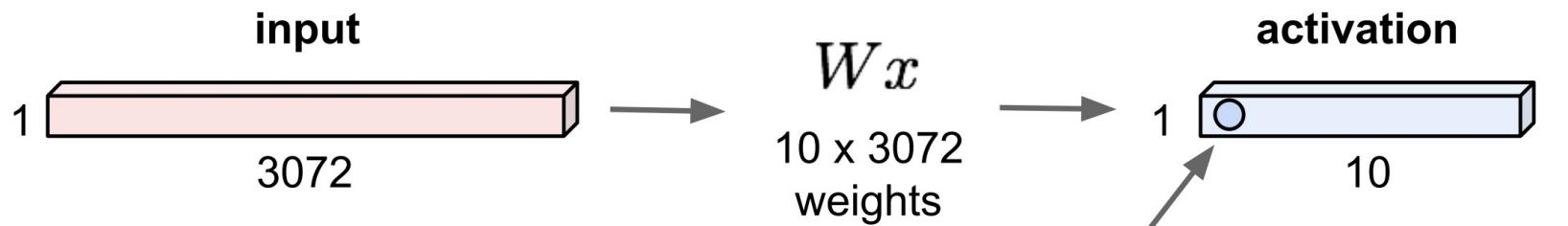
Backpropagation: cross-entropy loss

$$J(\theta) = \sum_i y^{(i)} \log(\hat{y}^{(i)})$$

# CNN Details

# Fully Connected Layer

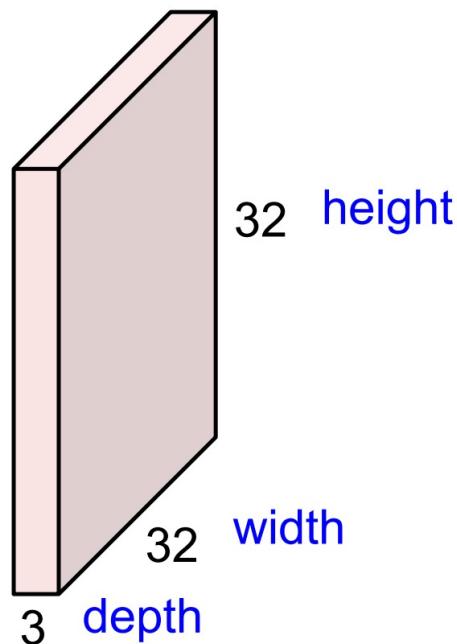
32x32x3 image -> stretch to 3072 x 1



**1 number:**  
the result of taking a dot product  
between a row of  $W$  and the input  
(a 3072-dimensional dot product)

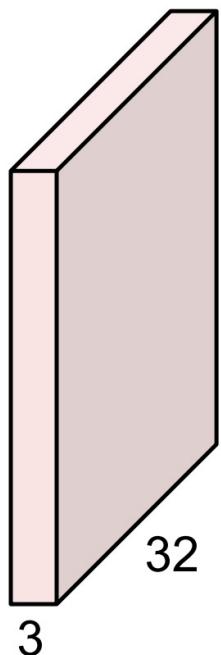
# Convolution Layer

32x32x3 image -> preserve spatial structure



# Convolution Layer

32x32x3 image



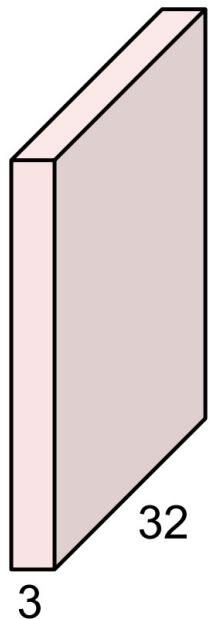
5x5x3 filter



**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

# Convolution Layer

32x32x3 image



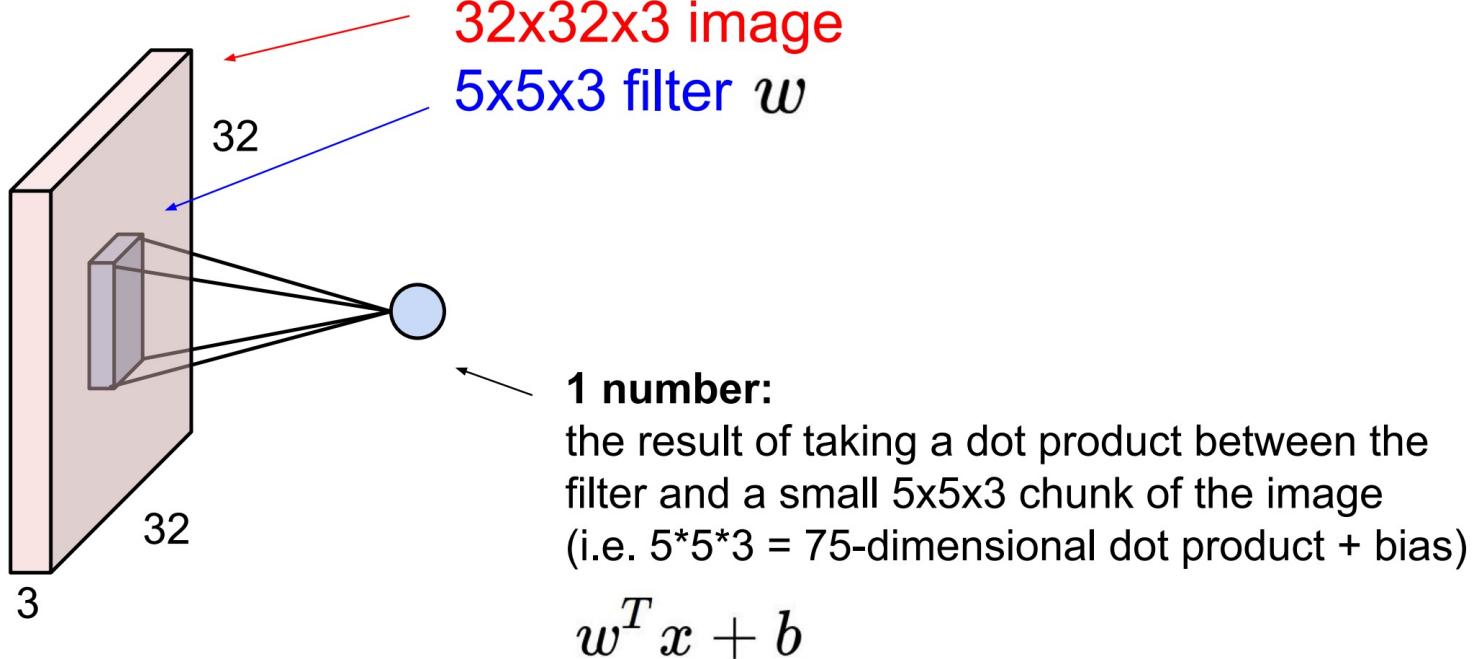
5x5x3 filter



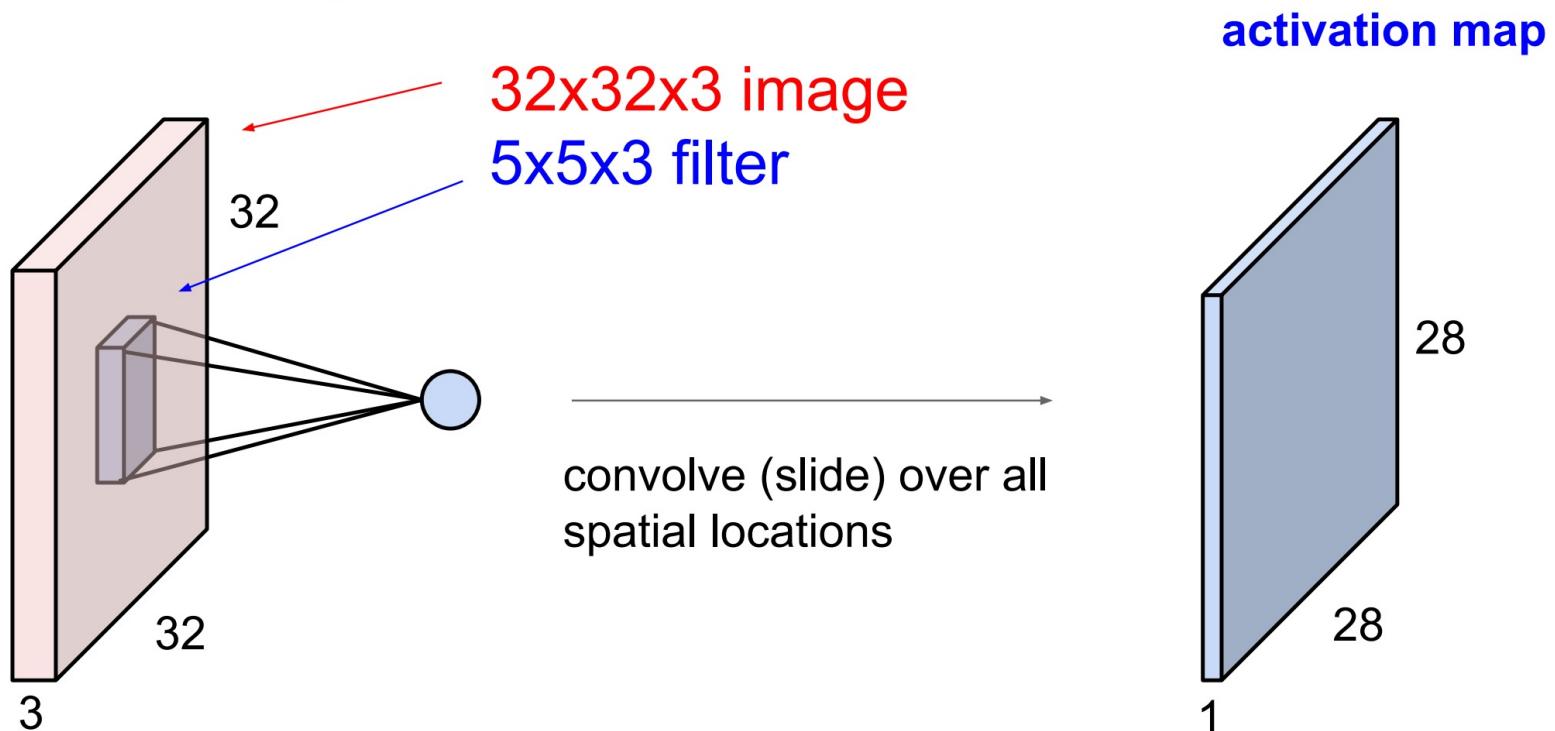
Filters always extend the full depth of the input volume

**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

# Convolution Layer

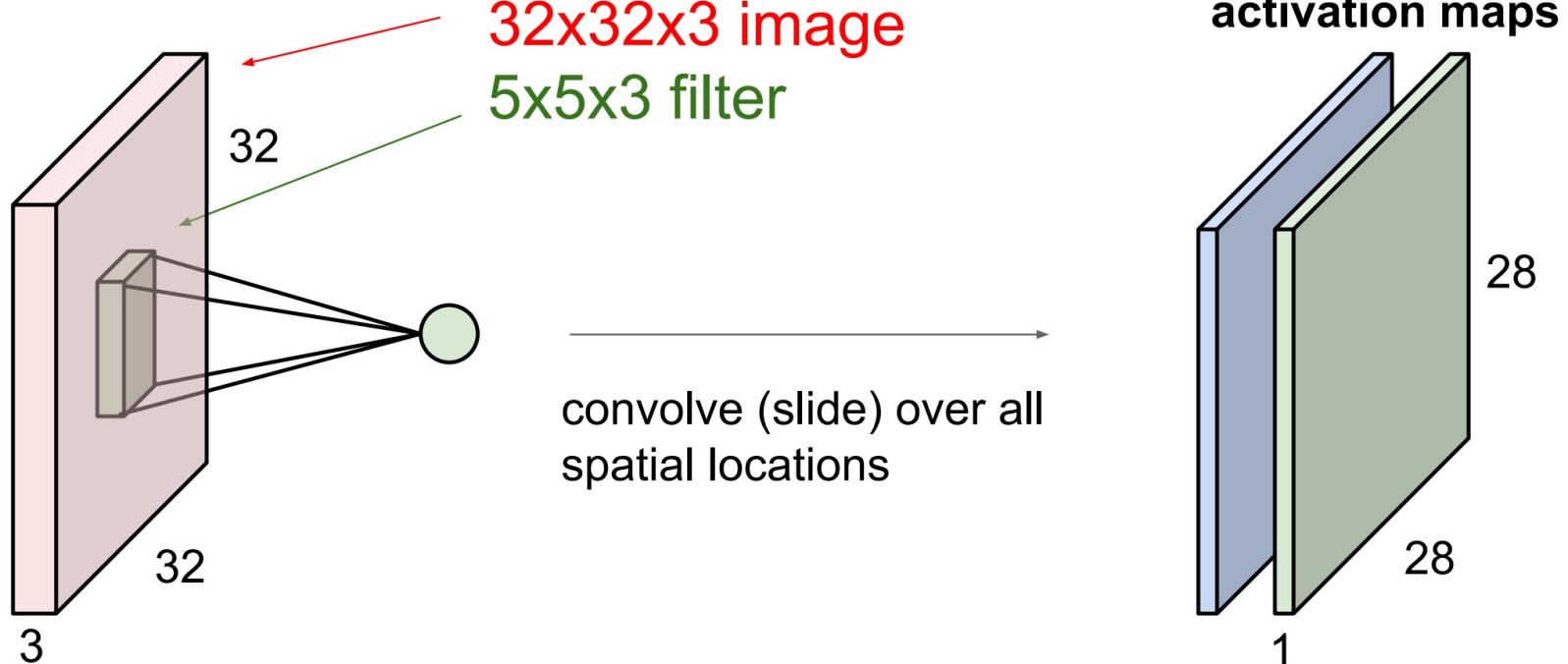


# Convolution Layer

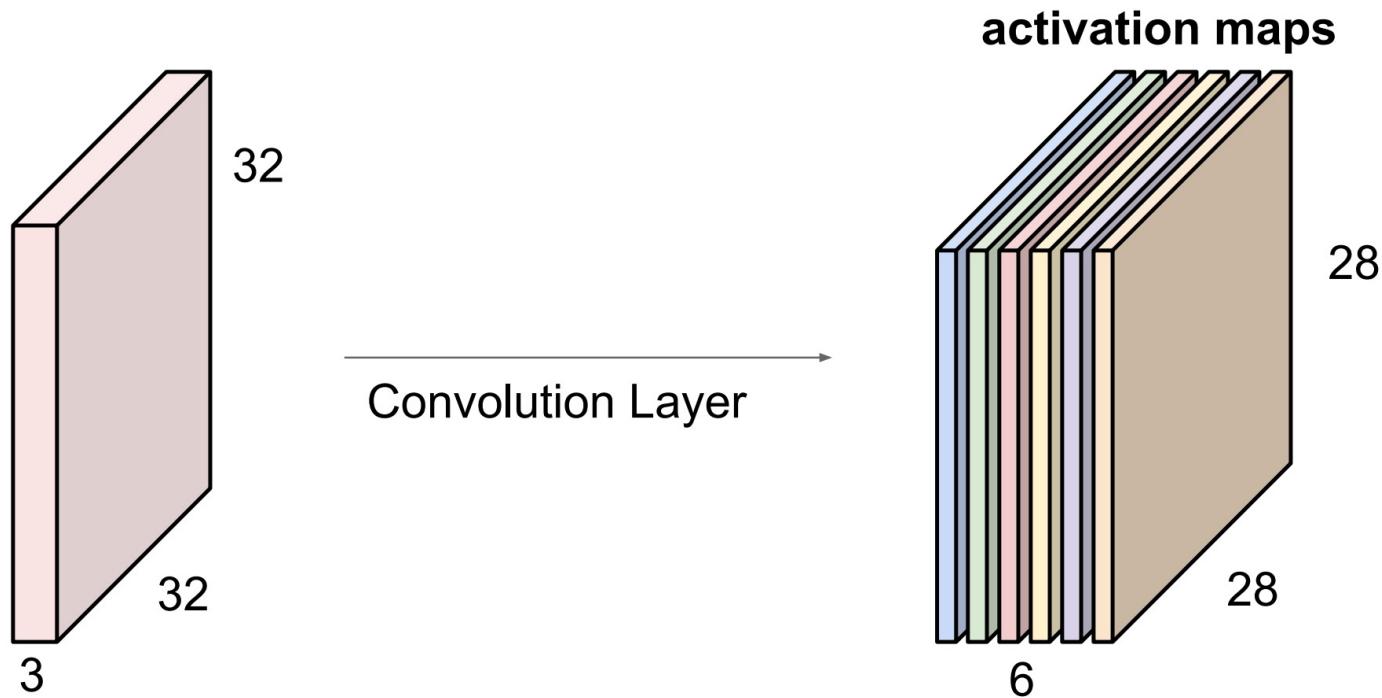


# Convolution Layer

consider a second, green filter

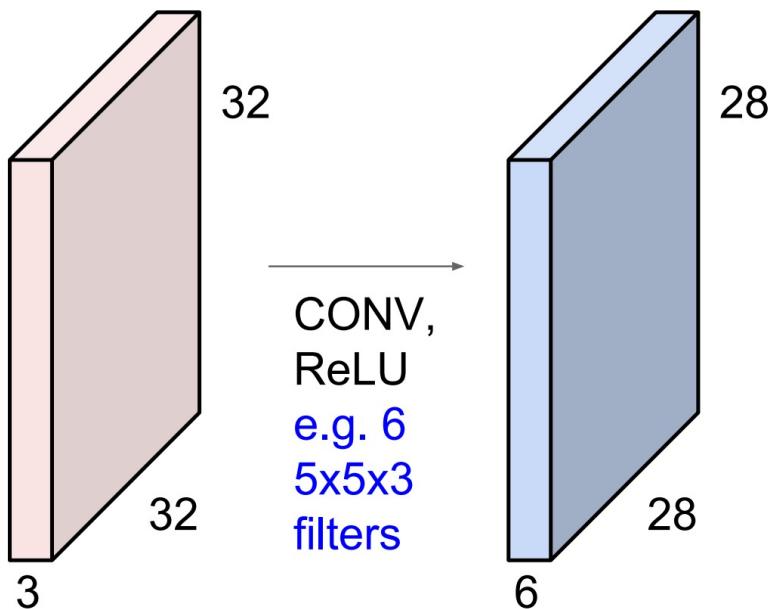


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

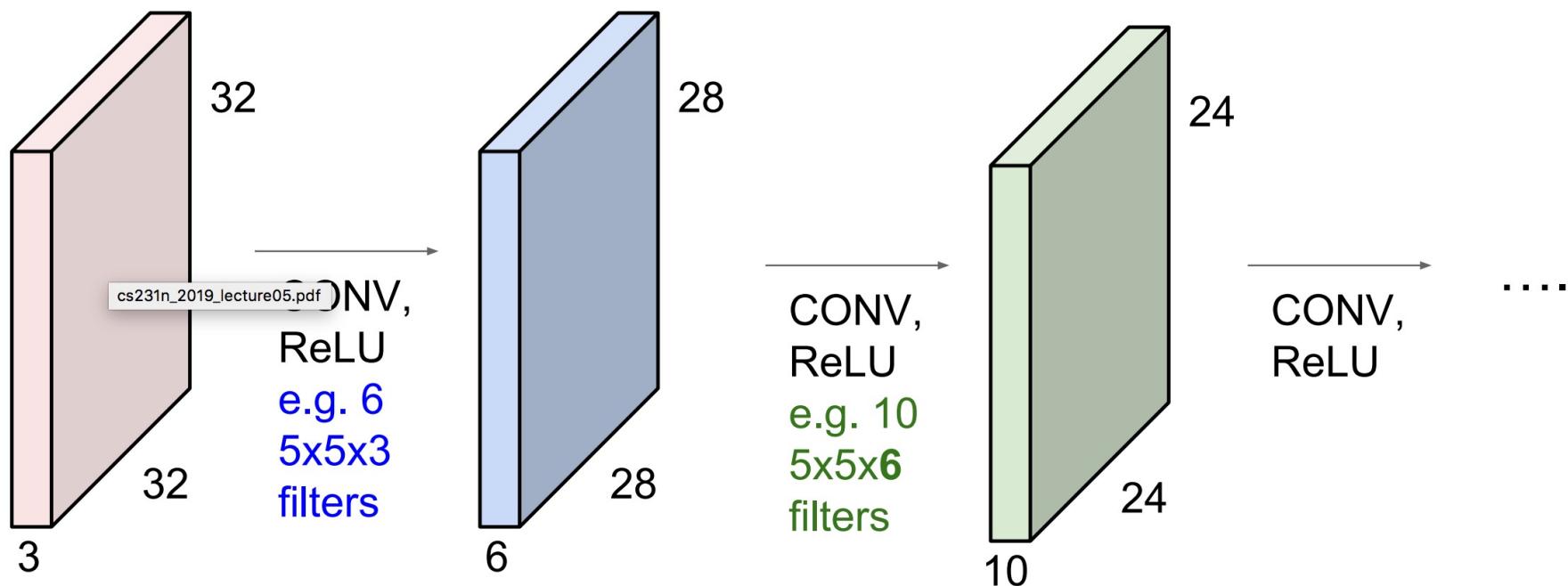


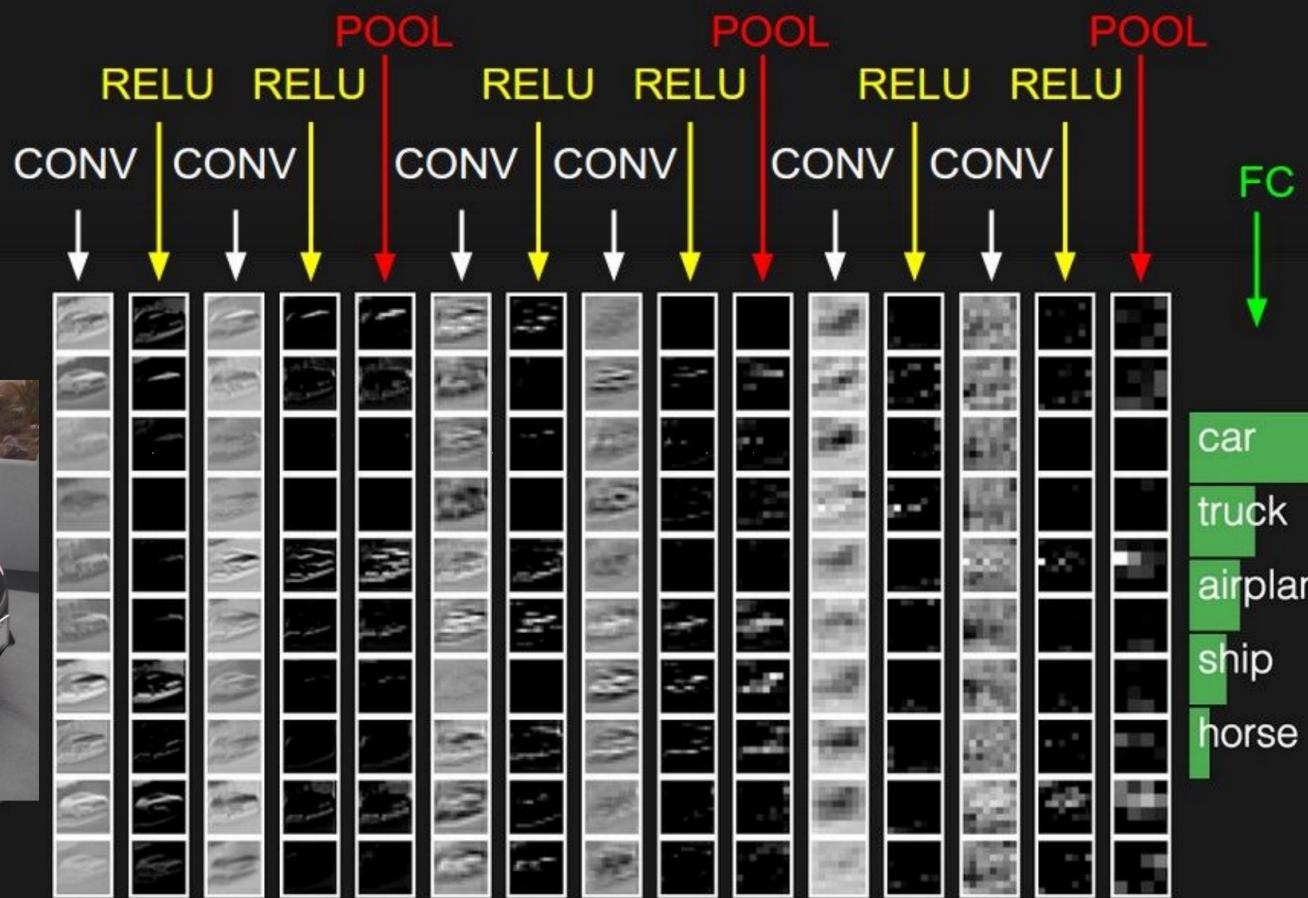
We stack these up to get a “new image” of size  $28 \times 28 \times 6$ !

**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions

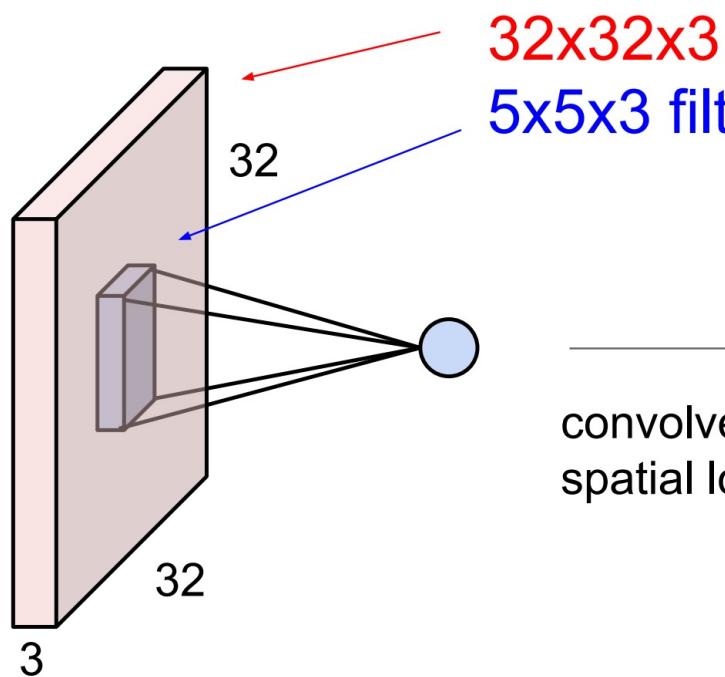


**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions

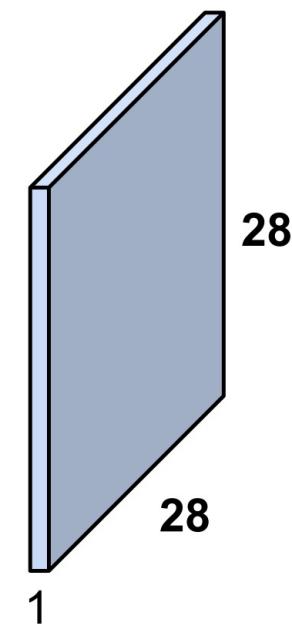




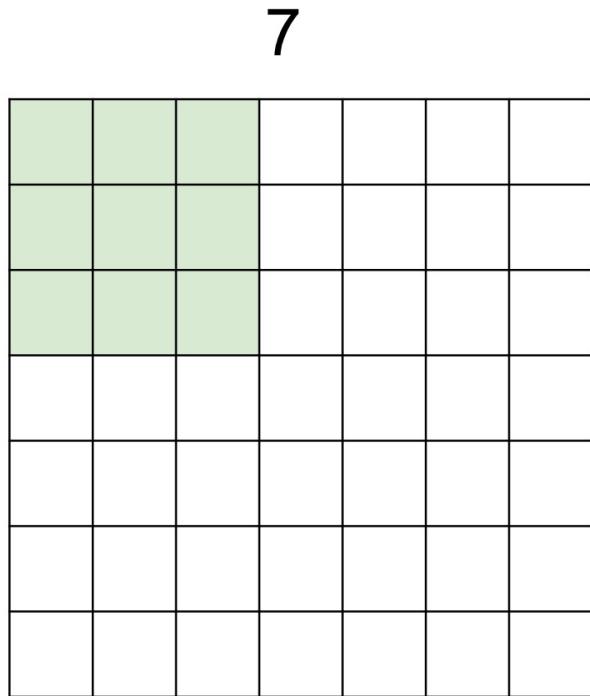
A closer look at spatial dimensions:



activation map



A closer look at spatial dimensions:

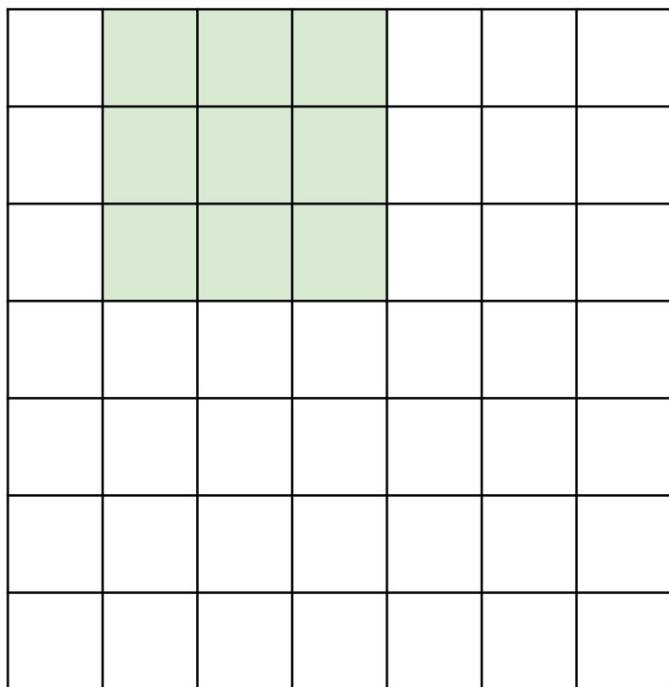


7x7 input (spatially)  
assume 3x3 filter

7

A closer look at spatial dimensions:

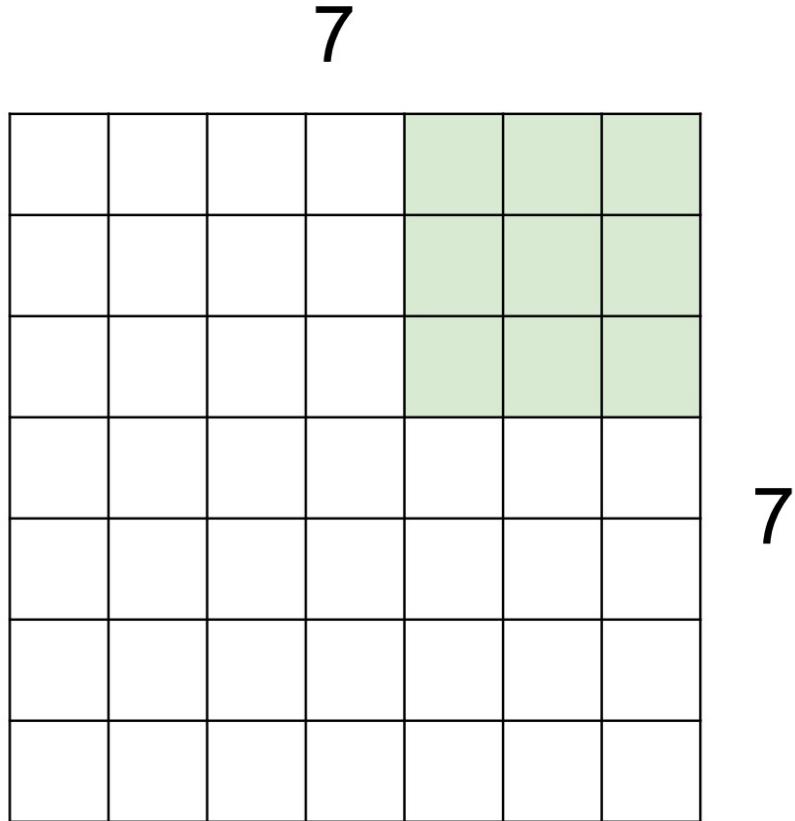
7



7x7 input (spatially)  
assume 3x3 filter

7

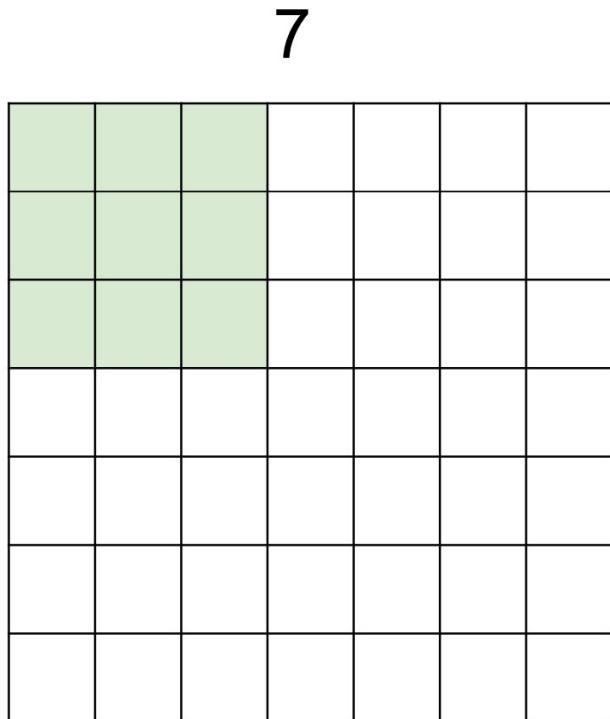
## A closer look at spatial dimensions:



7x7 input (spatially)  
assume 3x3 filter

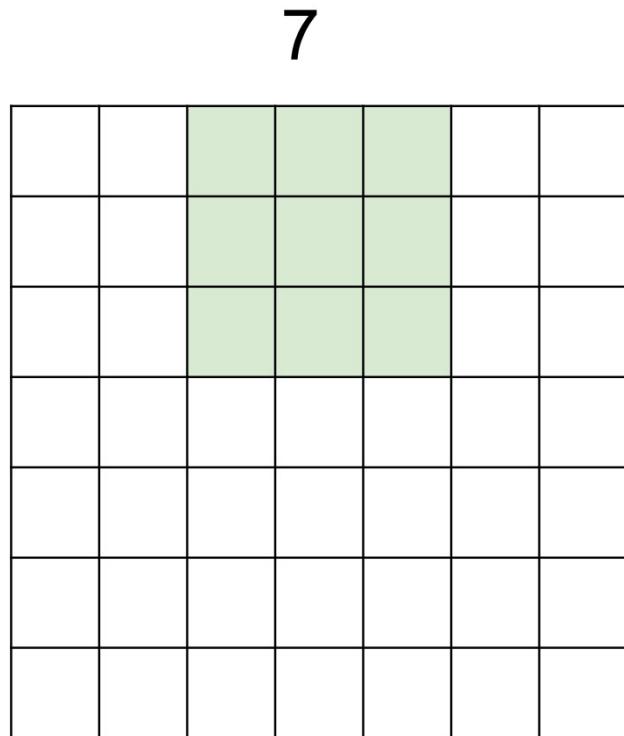
**=> 5x5 output**

A closer look at spatial dimensions:



7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 2**

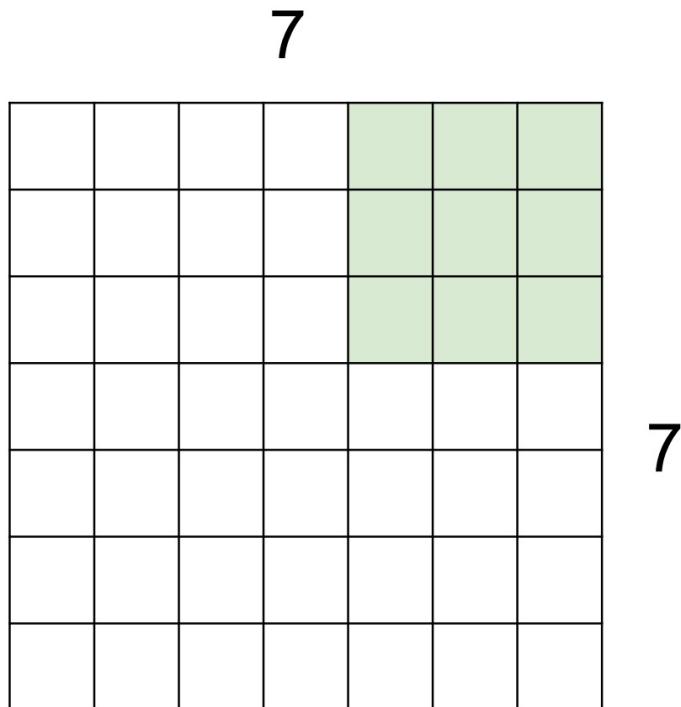
A closer look at spatial dimensions:



7

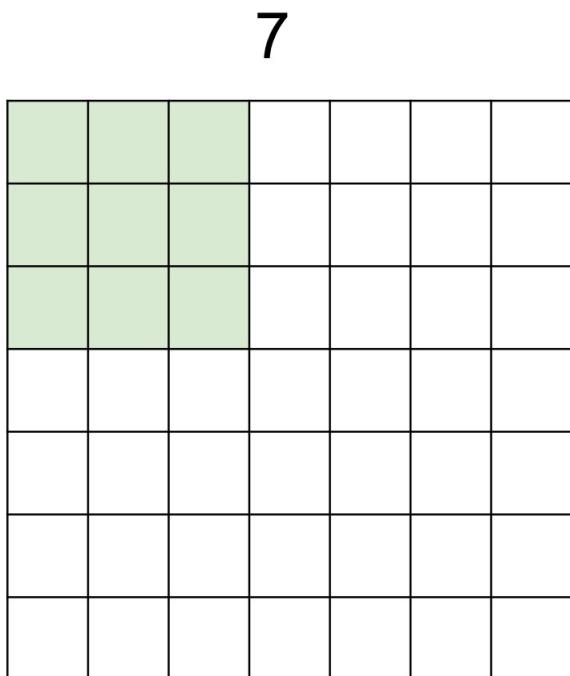
7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 2**

A closer look at spatial dimensions:



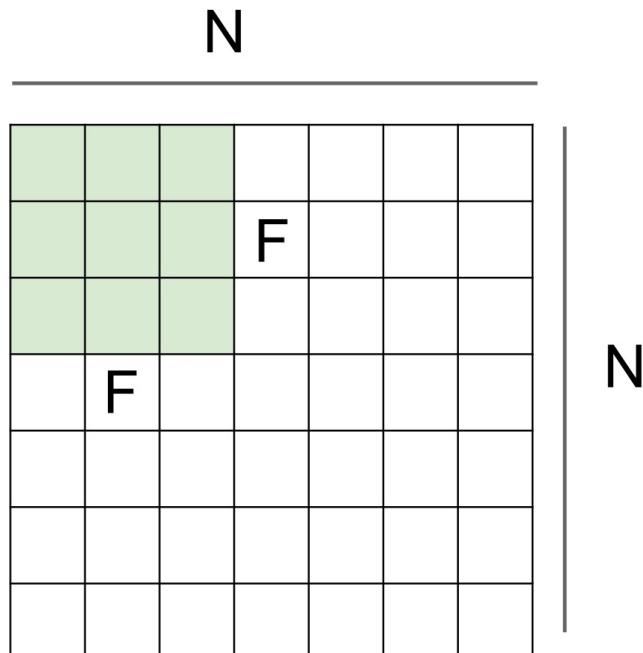
7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 2**  
**=> 3x3 output!**

A closer look at spatial dimensions:



7x7 input (spatially)  
assume 3x3 filter  
applied **with stride 3?**

**doesn't fit!**  
cannot apply 3x3 filter on  
7x7 input with stride 3.



Output size:  

$$\boxed{(N - F) / \text{stride}} + 1$$

e.g.  $N = 7$ ,  $F = 3$ :  
 stride 1  $\Rightarrow (7 - 3)/1 + 1 = 5$   
 stride 2  $\Rightarrow (7 - 3)/2 + 1 = 3$   
 stride 3  $\Rightarrow (7 - 3)/3 + 1 = 2.33 \therefore$

# In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

**3x3** filter, applied with **stride 1**

**pad with 1 pixel** border => what is the output?

(recall:)

$$(N - F) / \text{stride} + 1$$

# In practice: Common to zero pad the border

0	0	0	0	0	0		
0							
0							
0							
0							

e.g. input 7x7

**3x3 filter, applied with stride 1**

**pad with 1 pixel border => what is the output?**

**7x7 output!**

in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with  $(F-1)/2$ . (will preserve size spatially)

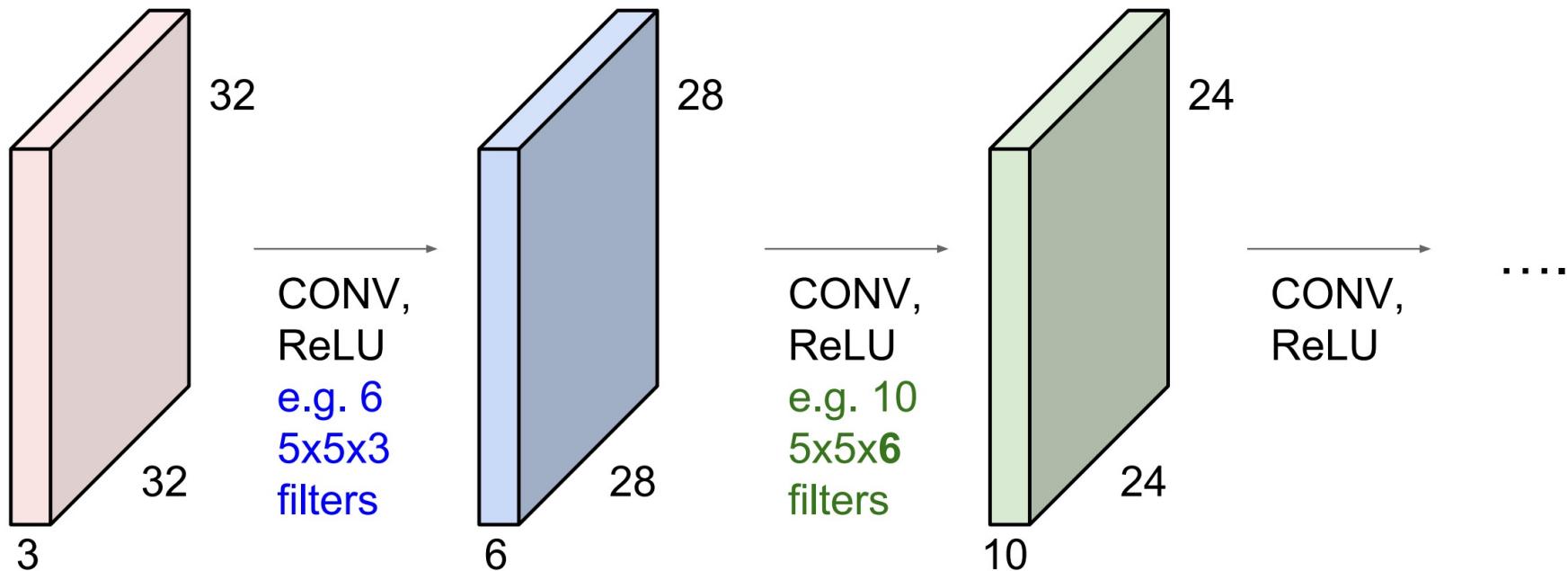
e.g.  $F = 3 \Rightarrow$  zero pad with 1

$F = 5 \Rightarrow$  zero pad with 2

$F = 7 \Rightarrow$  zero pad with 3

## Remember back to...

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!  
(32  $\rightarrow$  28  $\rightarrow$  24 ...). Shrinking too fast is not good, doesn't work well.



# Quiz

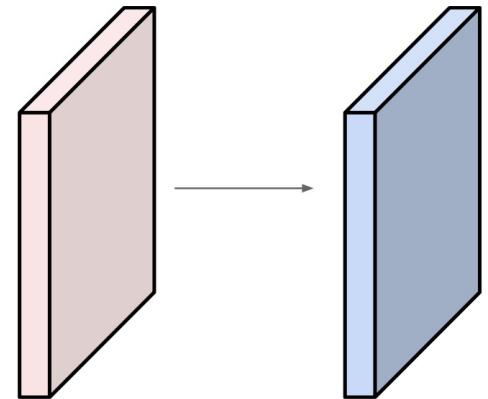
Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size: ?

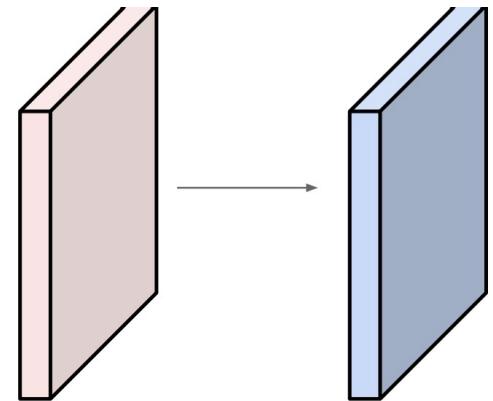
Number of parameters in this layer?



Examples time:

Input volume: **32x32x3**

10 **5x5** filters with stride 1, pad 2



Output volume size:

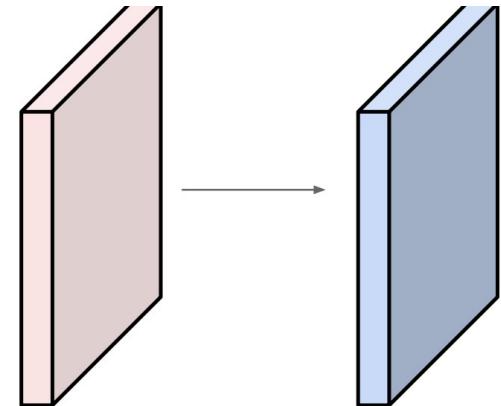
$(32+2*2-5)/1+1 = 32$  spatially, so

**32x32x10**

Examples time:

Input volume: **32x32x3**

10 **5x5** filters with stride 1, pad 2



Number of parameters in this layer?

each filter has  $5*5*3 + 1 = 76$  params      (+1 for bias)

$$\Rightarrow 76*10 = 760$$

**Summary.** To summarize, the Conv Layer:

- Accepts a volume of size  $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters  $K$ ,
  - their spatial extent  $F$ ,
  - the stride  $S$ ,
  - the amount of zero padding  $P$ .
- Produces a volume of size  $W_2 \times H_2 \times D_2$  where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$  (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces  $F \cdot F \cdot D_1$  weights per filter, for a total of  $(F \cdot F \cdot D_1) \cdot K$  weights and  $K$  biases.
- In the output volume, the  $d$ -th depth slice (of size  $W_2 \times H_2$ ) is the result of performing a valid convolution of the  $d$ -th filter over the input volume with a stride of  $S$ , and then offset by  $d$ -th bias.

Common settings:

- $K = (\text{powers of 2, e.g. } 32, 64, 128, 512)$
- $F = 3, S = 1, P = 1$
  - $F = 5, S = 1, P = 2$
  - $F = 5, S = 2, P = ? \text{ (whatever fits)}$
  - $F = 1, S = 1, P = 0$

```

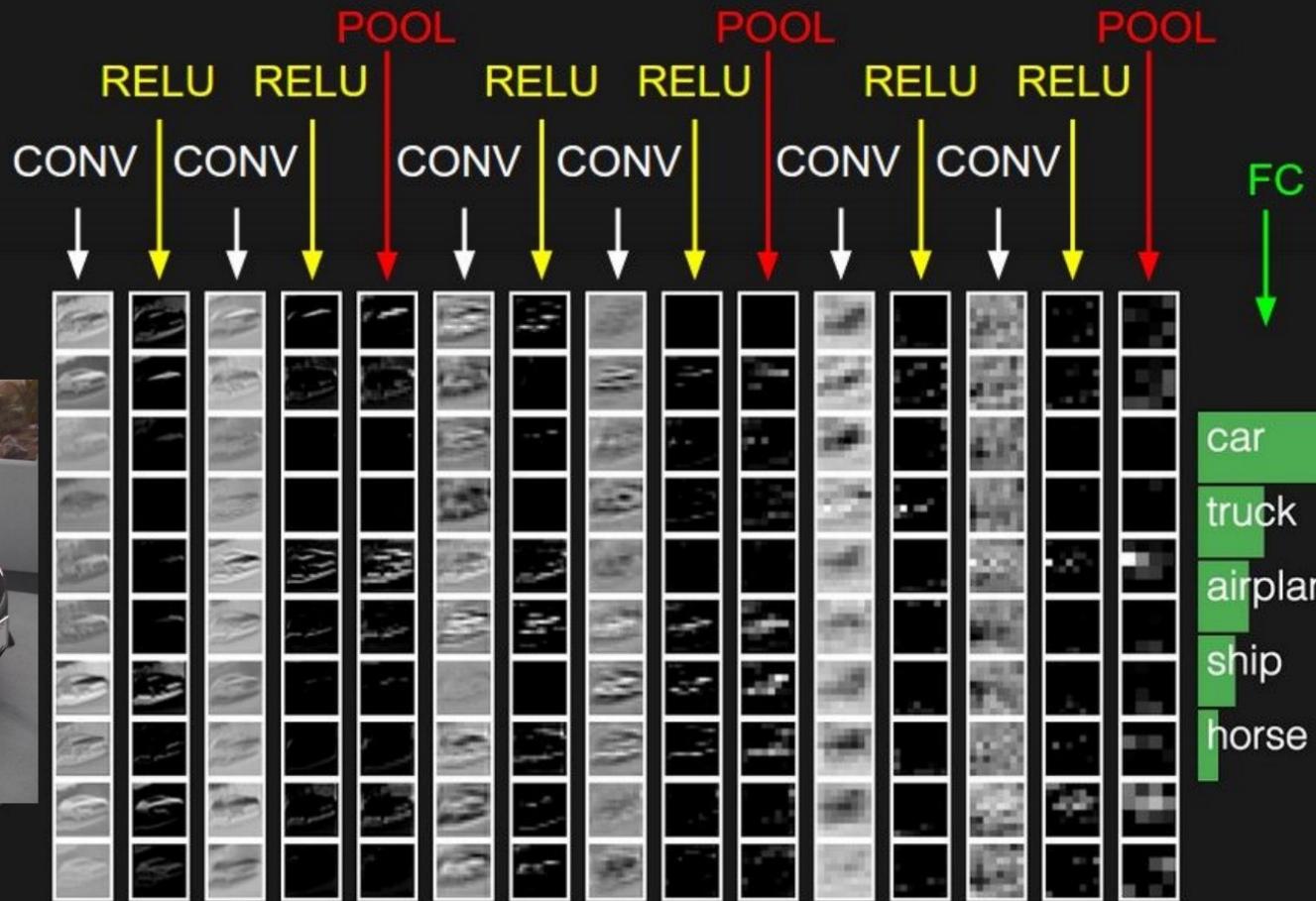
tf.keras.layers.Conv2D(
    filters, kernel_size, strides=(1, 1), padding='valid', data_format=None,
    dilation_rate=(1, 1), activation=None, use_bias=True,
    kernel_initializer='glorot_uniform', bias_initializer='zeros',
    kernel_regularizer=None, bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None, bias_constraint=None, **kwargs
)

```

Arguments:

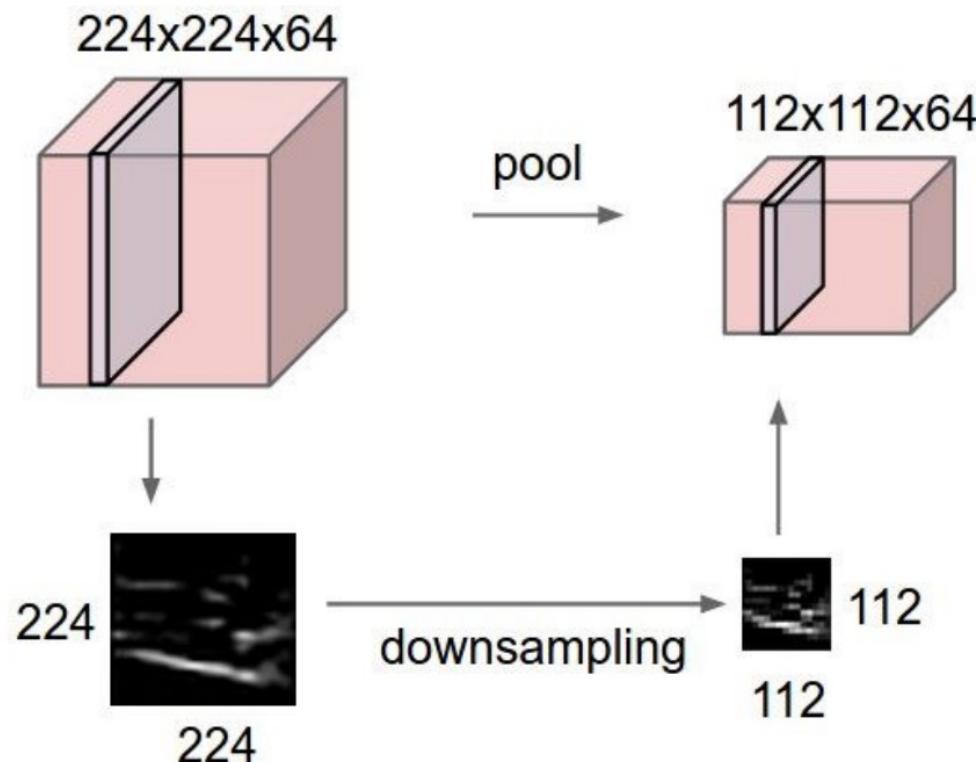
- `filters` : Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- `kernel_size` : An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- `strides` : An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- `padding` : one of "valid" or "same" (case-insensitive).
- `data_format` : A string, one of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape (batch, height, width, channels) while `channels_first` corresponds to inputs with shape (batch, channels, height, width). It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels\_last".
- `dilation_rate` : an integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any `dilation_rate` value != 1 is incompatible with specifying any stride value != 1.
- `activation` : Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation:  $a(x) = x$ ).
- `use_bias` : Boolean, whether the layer uses a bias vector.
- `kernel_initializer` : Initializer for the `kernel` weights matrix.
- `bias_initializer` : Initializer for the bias vector.
- `kernel_regularizer` : Regularizer function applied to the `kernel` weights matrix.
- `bias_regularizer` : Regularizer function applied to the bias vector.
- `activity_regularizer` : Regularizer function applied to the output of the layer (its "activation").
- `kernel_constraint` : Constraint function applied to the `kernel` matrix.
- `bias_constraint` : Constraint function applied to the bias vector.

[https://www.tensorflow.org/api\\_docs/python/tf/keras/layers/Conv2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D)



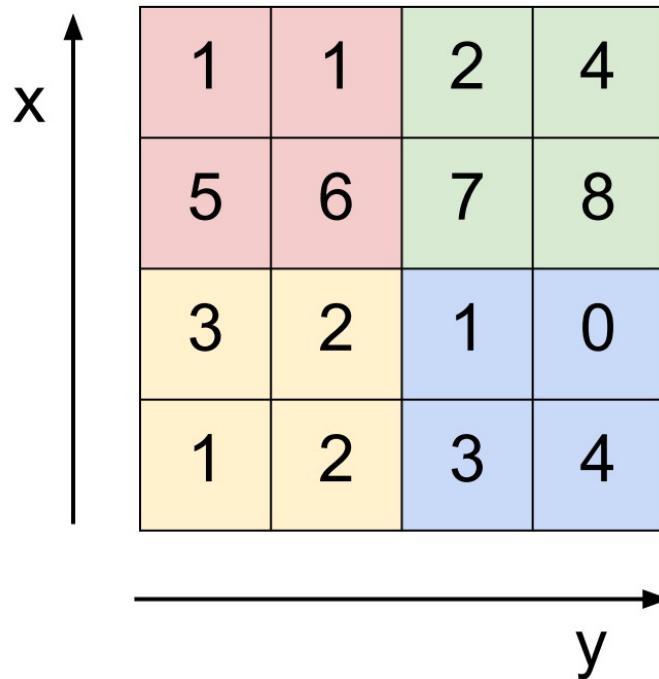
# Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



# MAX POOLING

Single depth slice



max pool with 2x2 filters  
and stride 2

6	8
3	4

## Common settings:

- Accepts a volume of size  $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent  $F$ ,
  - the stride  $S$ ,
- Produces a volume of size  $W_2 \times H_2 \times D_2$  where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

$F = 2, S = 2$

$F = 3, S = 2$

```
tf.keras.layers.MaxPool2D(  
    pool_size=(2, 2), strides=None, padding='valid',  
    data_format=None, **kwargs  
)
```

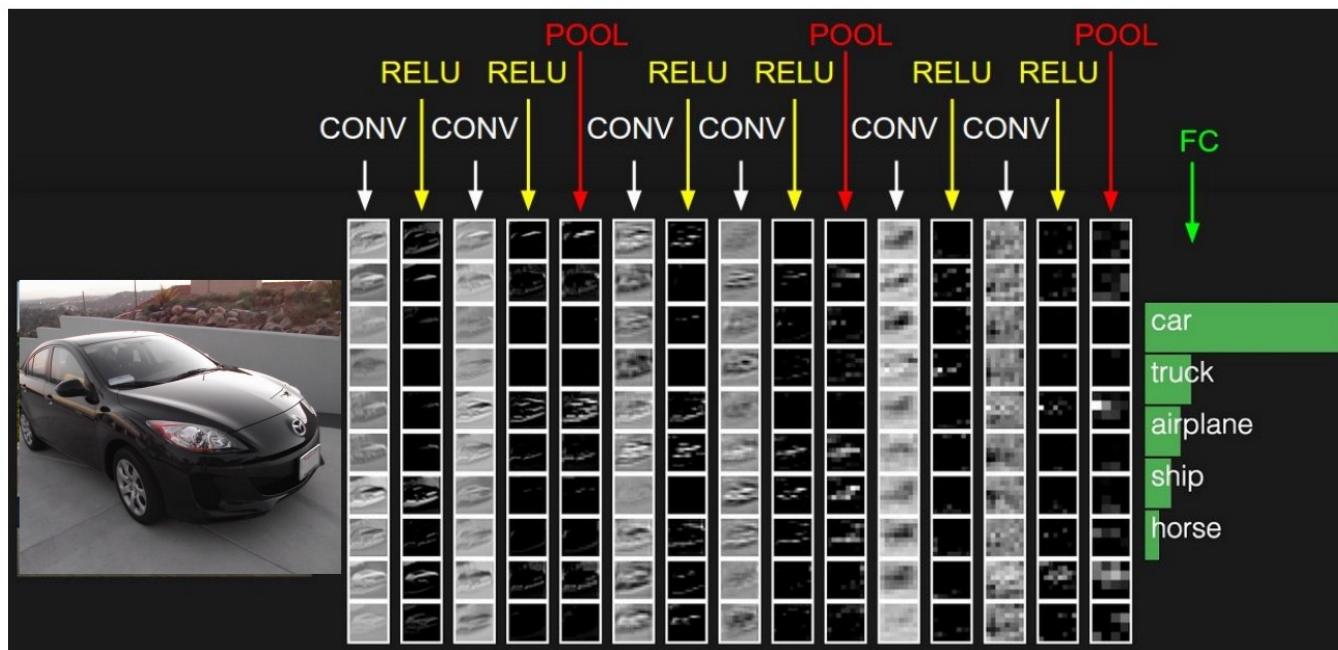
Arguments:

- `pool_size` : integer or tuple of 2 integers, factors by which to downscale (vertical, horizontal). `(2, 2)` will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions.
- `strides` : Integer, tuple of 2 integers, or None. Strides values. If None, it will default to `pool_size`.
- `padding` : One of `"valid"` or `"same"` (case-insensitive).
- `data_format` : A string, one of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape `(batch, height, width, channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, height, width)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be `"channels_last"`.

[https://www.tensorflow.org/api\\_docs/python/tf/keras/layers/MaxPool2D](https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D)

# Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



**Table 4.1 Choosing the right last-layer activation and loss function for your model**

Problem type	Last-layer activation	Loss function
Binary classification	sigmoid	binary_crossentropy
Multiclass, single-label classification	softmax	categorical_crossentropy
Multiclass, multilabel classification	sigmoid	binary_crossentropy
Regression to arbitrary values	None	mse
Regression to values between 0 and 1	sigmoid	mse or binary_crossentropy

Source: Deep Learning with Python

# CNN Code Examples

EX 1: [https://colab.research.google.com/drive/1zxMGI74kmZk2wohjUw\\_oAevRWiOGBaIF](https://colab.research.google.com/drive/1zxMGI74kmZk2wohjUw_oAevRWiOGBaIF)

EX 2: <https://colab.research.google.com/drive/1KBfr0SDpsQMvfaxzglIN5uwxnrUDjl4p>

Ex 3: <https://colab.research.google.com/drive/1yroklTQuaejg95ALMrg9IcZBM8MpjTAp>

# Additional References

---

- Chapter 5 of [T2]